Tessera: User-Defined Compiler Optimizations for High-Performance Scientific Computing

Jessica Cotturone William Moses (advisor)

University of Illinois Urbana-Champaign

Motivation

- General-purpose compilers have no knowledge of domain-specific properties.
- Semantic meaning is lost by the time the code is lowered to an intermediate representation (IR).

opaque pointer

```
%4 = call ptr @gram_matrix(ptr %0, double 1.0e-03)
call void @lu_solve(ptr %4, ptr %1, ptr %2)
```

opaque library call

- The compiler can't see through the pointer or library call to recognize that the constructed Gram matrix is symmetric positive definite (SPD) and replace the LU solve with a faster Cholesky solve.
- Manual optimization is tedious, error-prone, and must be repeated in every application.
- **Goal: provide a unified approach to domain-specific compiler optimization where experts specify high-level properties and optimization opportunities that the compiler exploits wherever possible.**

Approach

- Domain scientists **annotate library functions** and **define optimizations** on these functions, which are shipped with the library and imported into any application that uses it.
- The compiler, built on Clang and MLIR, **parses** and **propagates** this semantic information, then uses pattern matching to automatically rewrite application code.
- This approach is designed to generalize across domains and allow optimizations from different scientific subfields to interoperate.
- In the example below, the compiler now knows the Gram matrix is SPD and replaces the LU solve with a faster Cholesky solve.

lib.h

```
__attribute__((tessera_guarantees("SPD")))
Matrix gram_matrix(Mat A, double lambda);

__attribute__((tessera_op("lu_solve(A, b, x)")))
void lu_solve(Mat A, Vec b, Vec x);

__attribute__((tessera_op("cholesky_solve(A, b, x)")))
void cholesky_solve(Mat A, Vec b, Vec x);
```

shipped with the library

tesseraOptimizations.h

```
#pragma optimize "if SPD(A), lu_solve(A, b, x) ->
cholesky_solve(A, b, x)"
```

application.cpp

```
Matrix G = gram_matrix(A);
lu_solve(G, b, x);
```

→

```
Matrix G = gram_matrix(A);
cholesky_solve(G, b, x);
```

Tessera recognizes the property and automatically rewrites the code

Annotations

- Annotations provide a **lightweight, non-intrusive** way for domain experts to convey semantic information to the compiler.
- Five main annotation types (below) allow library experts and application writers to indicate properties of functions' inputs, outputs, and behaviors.

Annotation	Meaning	Example
`tessera_op`	Operation has a custom optimization	LU solve can be rewritten
`tessera_guarantees`	Result has a certain property	Regularized Gram matrix is SPD
`tessera_assumes`	Input has a certain property	Cholesky solve requires SPD input
`tessera_preserves`	Output inherits property of input	Sum of symmetric matrices is symmetric
`tessera_readonly`	Reads its arguments, never writes them	Matvec only reads its matrix

Optimization Rules

- Optimization rules express rewrites that library experts assert are valid, giving the compiler permission to apply them.
- Tessera annotations can be added to each library's header file, and optimizations from all libraries **can be combined** to optimize an application.
- Conditional optimizations are checked at compile time when possible, and otherwise are checked at runtime.
- Tessera recognizes and rewrites call patterns involving the MFEM finite element library [1] and the MPFR library for arbitrary-precision floating-point arithmetic [2].

Simplify call pattern

```
mfem.MultABt(F,F,B)
```

→

```
mfem.MultAAt(F,B)
```

Conditional strength reduction

```
mpfr.pow(rop,x,y,rnd)
```

→

```
mpfr.sqrt(rop,x,rnd)
```

```
if mpfr.cmp_d(y,0.5)==0
```

Nested call strength reduction

```
mpfr.div(rop,x,mpfr.set_ui_
2exp(c,1,k,rnd),rnd)
```

→

```
mpfr.div_2si(rop,x,k,rnd)
```

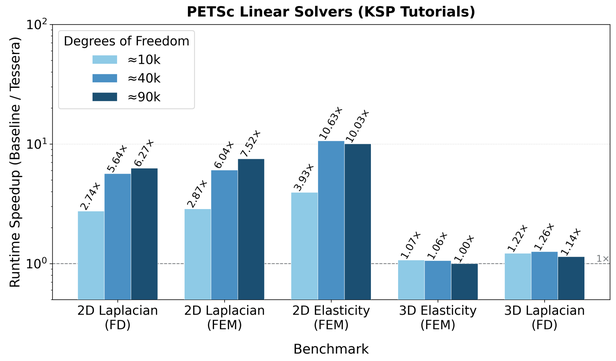
Key Takeaways

★ Tessera lets domain scientists write **custom optimizations** on library functions based on mathematical and structural properties, then propagates these properties and **automatically applies** the optimizations wherever they are valid.

★ By preserving semantic information through annotations, Tessera bridges the gap between **domain-specific expertise** and **general-purpose compiler infrastructure**, allowing compilers to optimize over library calls they would normally see as black boxes.

Results: PETSc Solver Selection

- The PETSc library [3] provides iterative methods for solving large, sparse linear systems, including Generalized Minimal Residual (GMRES) and Conjugate Gradient (CG).
- CG is typically much cheaper than GMRES, but can only be applied when the matrix is known to be SPD.
- Using only a few annotations that are **reusable across all benchmarks**, Tessera recognizes that the matrices are SPD and selects CG over GMRES.
- This rewrite achieves up to a **10.63× speedup** on a 2D elasticity benchmark.
- Speedups compare each benchmark compiled with and without Tessera annotations.



FD = finite difference; FEM = finite element method

Conclusions

- Tessera lets library experts declare properties and optimizations once in a library header, and the compiler **applies them automatically** at every matching call site in any application, rather than requiring manual optimization in each one.
- Automated SPD-based solver selection in PETSc benchmarks achieves up to a **10.63× speedup** with only a few reusable annotations.
- Tessera recognizes patterns and suboptimal calls to MFEM and MPFR, then rewrites them using functional and strength-reduction rules.
- **Future work:** letting the compiler choose when to apply rewrites using a cost model and extending Tessera to more HPC applications.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation under Grant No. 2346519, in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and in part by the U.S. Department of Energy, National Nuclear Security Administration under Award Number DE-NA0004266. This material is also based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Department of Energy Computational Science Graduate Fellowship under Award Number DE-SC0026617.

References

[1] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny, V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, and S. Zampini, "MFEM: A modular finite element methods library," *Computers & Mathematics with Applications*, vol. 81, pp. 42–74, 2021.

[2] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," *ACM Transactions on Mathematical Software*, vol. 33, no. 2, 2007.

[3] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

Contact: jac18@illinois.edu